Task Allocation in Industrial Edge Networks with Particle Swarm Optimization and Deep Reinforcement Learning

Philippe Buschmann  
philippe.buschmann@siemens.com  
Siemens AG  
Munich, Bavaria, Germany

Mostafa H. M. Shorim  
shorim@net.in.tum.de  
Siemens AG  
Munich, Bavaria, Germany

Max Helm  
helm@net.in.tum.de  
Technical University of Munich  
Garching, Bavaria, Germany

Arne Bröring  
arne.broering@siemens.com  
Siemens AG  
Munich, Bavaria, Germany

Georg Carle  
carle@net.in.tum.de  
Technical University of Munich  
Garching, Bavaria, Germany

ABSTRACT

To avoid the disadvantages of a cloud-centric infrastructure, next-generation industrial scenarios focus on using distributed edge networks. Task allocation in distributed edge networks with regards to minimizing the energy consumption is NP-hard and requires considerable computational effort to obtain optimal results with conventional algorithms like Integer Linear Programming (ILP). Today’s prevailing cloud-centric Internet of Things (IoT) model has limitations [10], e.g., (i) unreliable cloud connectivity, (ii) limited bandwidth, and (iii) high round-trip times. Therefore, next-generation IoT applications, such as autonomous guided vehicles, AR/VR, or industrial robots/drones, require advanced IoT environments that comprise heterogeneous devices, which collaboratively execute these IoT applications [15].

In this work, we focus on the industrial edge computing infrastructure as advanced IoT environment. Users of industrial edge networks can scale resources horizontally on multiple local network nodes. This allows moving workflows and applications from the cloud to local nodes which can reduce latency and can increase throughput and reliability [2]. Further, this can enhance the privacy and security of the workflow [5].

One disadvantage of the industrial edge is the complexity of placing workflow on a distributed network. A workflow can be constrained by Quality of Service (QoS) objectives like latency or energy-consumption and consists of one or multiple tasks that communicate with each other. Thus, a random placement of tasks in a workflow on network nodes might not satisfy the QoS constraints of the workflow. To optimize the QoS constraints while adhering to the physical boundaries of the network, it is necessary to design specific algorithms for solving the task allocation problem. This problem is known as NP-hard [1] and can be solved by either optimal or heuristic approaches.

Optimal Approach To allocate tasks optimally on the nodes of an edge network, we can use well-known optimization techniques like ILP, and Optimization Modulo Theories (OMT). These methods achieve an optimal solution [1, 19], but scale poorly in large networks with many applications [19]. For example, the ILP model in [19] can take up to a week of computational time to find the
optimal allocation of 50 tasks on 20 nodes in a network with the objective of minimizing the total energy-consumption [19].

**Heuristic Approach** Heuristics and meta-heuristics only approximate the optimal solution and usually scale better than optimal approaches [20, 22]. Some well-known heuristics are the Genetic Algorithm (GA) and PSO [20, 22]. Since these algorithms only approximate the optimal solution, they may not satisfy the QoS constraints perfectly or may perform worse than the optimal solution, which is generally known as optimality gap [1].

In this work, we investigate both orchestration methods for optimal and approximated workflow allocation. We extend an existing ILP model to optimize the allocation of multiple workflows with regards to the capabilities of task and nodes and we implement and evaluate a PSO and a DRL approach for the approximated allocation of tasks. Since the energy consumption of networked IoT devices and its environmental impact gains more awareness [16], our objective is to minimize the energy consumption of the overall network. In other words, we focus on the energy consumption of all allocated and executed tasks on the devices and their energy consumption for communication in between.

The main contributions are as follows:

- We extend an ILP model (Section 4) which allocates tasks of multiple workflows optimally on a network with energy consumption as a cost function. In contrast to previous work [19], this eliminates the bias towards previously allocated workflows when placing one workflow at a time. Furthermore, we implement a constraint which limits the allocation of tasks on nodes based on their capabilities.
- We define and implement a PSO approach for task allocation. Then, we evaluate the approach against the ILP. We show that PSO outperforms the previously proposed heuristic in [19] but with the trade-off of time consumption.
- We implement, train, and evaluate a DRL model using Proximal Policy Optimization (PPO). The trained model has a lower optimality gap in comparison to PSO and the extended version of the heuristic in [19].

In Section 2, we define the task allocation problem in edge computing networks and explain some optimization approaches in detail. Next, we identify similarities and differences in related work in Section 3. In Section 4, we describe our approaches and techniques in our methodology and evaluate them in Section 5. Last, we conclude our work in Section 6.

## 2 BACKGROUND

First, we define the task allocation problem used in this work. To optimize this problem, we use Integer Linear Programming (ILP), Particle Swarm Optimization (PSO) and Deep Reinforcement Learning (DRL). We introduce each method and algorithm and explain the approaches to find a solution for the optimization problem. We start with a description of ILP problems and explain the algorithms which allow to solve ILP problems. Then, we describe the PSO algorithm. Last, we give an overview of DRL and Proximal Policy Optimization (PPO).

### 2.1 Problem Definition

In this work, we focus on optimizing the allocation of multiple workflows on a network. Therefore, we define all parts of this optimization problem. We visualize the problem in Figure 1. In the context of this work, a workflow is a connected and directed acyclic graph with nodes that we call tasks. A task can be e.g., an application, a docker container or a micro-service. In our model, tasks cannot be split across multiple nodes in a network, run indefinitely and do not terminate. They require a static amount of resources like Central Processing Unit (CPU) cycles, Random Access Memory (RAM), and storage, which is visualized as squares. Tasks communicate with other tasks in the workflow and transmit data at a specific rate (transmission output) to other tasks. A task can require a specific capability e.g., a graphical user interface or access to sensors and actuators which further limits the allocation possibilities in a network.

Another component in our model is the network which is a bidirectional and connected graph. The network consists of one or more nodes which are connected to other nodes. Each node provides resources like CPU, RAM and storage which can be zero if no resources are available anymore. In Figure 1, Node 4 accommodates Task 3 and has no resources left for other tasks. Other nodes can still provide all their resources. Each node can offer the capabilities which can be used by tasks. These can be e.g., a temperature sensor, a specific machine or API which is required by a task.

Our first objective is to allocate tasks of multiple workflows to the nodes of the network. Therefore, tasks can only be placed on nodes with enough resources to accommodate the task. One node can accommodate multiple tasks if enough resources are available on that node. Task can only be allocated and executed on nodes that provide the required capabilities.

Our second objective is to allocate tasks on a network with regards to a specific cost function. In general, this cost function can be any QoS constraint. Like [19], we focus on minimizing the energy consumption of the network. Thus, we define the total energy consumption as the sum of the utilization of nodes (if they accommodate tasks) and the utilization of connections between the nodes (if used to transmit data between tasks). To minimize the energy consumption, we can place tasks on nodes that use their resources more efficiently and allocate tasks on nodes close to each other. We show this possibility in Figure 1, where Task 4 can be placed on either Node 2 or 3. In this case, Node 3 offers a lower amount of energy consumption per resource.

As summary, we need to allocate tasks on a network of nodes so that we use the least amount of energy in sum over all devices and network connections while finding a feasible solution with regards to resource consumption and capabilities.

### 2.2 Integer Linear Programming

In this work, we define the problem of Section 2.1 as ILP problem. In general, Linear Programming (LP), Integer Linear Programming (ILP), and Mixed Integer Linear Programming (MILLP) are methods to formulate an optimization problem. If it is possible to formulate the problem as an (mixed) (integer) linear program, we can apply
Workflows

\[
\begin{align*}
\text{Workflows} & \\
\text{Workflow 1} & \\
\text{Task 0} & \rightarrow \text{Task 1} & \rightarrow \text{Task 2} & \rightarrow \text{Task 3} & \rightarrow \text{Task 4} & \rightarrow \ldots \\
\end{align*}
\]

\hspace{1cm}

Network

\[
\begin{align*}
\text{Network} & \\
\text{Node 0} & \rightarrow \text{Node 1} & \rightarrow \text{Node 2} & \rightarrow \text{Node 3} & \rightarrow \text{Node 4} & \rightarrow \text{Node 5} \\
\end{align*}
\]

\hspace{1cm}

Figure 1: Visualization of the task allocation problem

...
(\text{pbest}_i - x^i_t) \) introduces the direction towards the personal best value of the particle \text{pbest}_i multiplied with a random value \( r_1 \) and a time varying coefficient \( c_1 \). The third segment \( c_2 \times r_2 \times (\text{gbest} - x^i_t) \) introduces the direction towards the global best value of all particles multiplied with a random value \( r_2 \) and a time varying coefficient \( c_2 \). The time varying coefficients \( c_1 \) and \( c_2 \) influence the focus on the personal best values and on the global best values. We add the velocity \( \Delta x^i_{t+1} \) to the previous position \( x^i_t \) to get the new position \( x^i_{t+1} \). This position can then be used for the velocity in the next iteration.

The personal and global best value is calculated by the fitness function \( F(x) = f(x) \) with the position vector \( x \) as possible solution. \( f(x) \) is the objective function of the optimization problem which returns the value for solution \( x \). In addition to the objective function, we can introduce constraints to \( F(x) \) by using Deb’s approach [6] which is the most common approach in the literature [11]. We show this in Equation 5. We define the constraints as inequality constraints \( g_j(x) \geq 0 \) for \( j = 1, ..., n \) with \( n \) being the maximum number of all constraints. If the solution \( x \) does not violate any constraint \( (g_j(x) \geq 0) \), PSO returns the objective function \( f(x) \). If \( x \) violates at least one constraint, we return the worst objective value \( f_{\text{worst}} \) (e.g., the maximum energy consumed) plus the amount of how much constraints where violated \( (g_j(x)) \).

\[
F(x) = \begin{cases} 
  f(x), & \text{if } g_j(x) \geq 0, \forall j \in \{0, ..., n\} \\
  f_{\text{worst}} + \sum_{j=1}^{n} |g_j(x)|, & \text{otherwise} 
\end{cases} \tag{5}
\]

### 2.4 Deep Reinforcement Learning

Another more recent approach to solving optimization problem is DRL. We choose DRL because we can approach it similarly to PSO, we do not need labeled data as for supervised learning [4] and the neural network does not have to find a hidden structure as for unsupervised learning [14].

DRL can be split into agent, environment, action, observation and reward [14]. Figure 2 shows the dependencies between these entities. We initialize an environment with a state that can be observed. This observation is used by the Deep Neural Network (DNN) to return a probability distribution of actions. The policy selects one action which is then applied to the environment. Then, the agent can observe the environment again and use the policy to select the next action. This is repeated if no termination condition is met.

From the observation, we can calculate a reward value which can be used to train the DNN. We can calculate a reward by using e.g., the objective function of the ILP problem and/or the fitness function of the PSO algorithm. In general, a DRL agent should aim to receive the highest reward for its actions.

Selecting an action that yields the highest reward depends highly on the policy [14]. We use the PPO in this work. PPO was proposed by Schulman et al. [18] as a simpler alternative to Actor-Critic Experience Replay (ACER) [21] with similar performance which outperforms the synchronous Advantage Actor Critic (A2C) [17].

In this work, we use the implementation of the MaskablePPO, which disallows the use of invalid actions [9].

### 3 RELATED WORK

Our work builds upon the previous work of Seeger et al. [19]. Seeger et al. focuses on minimizing the energy consumption of a network by optimally allocating the tasks of a workflow on the edge. For that, they extend the framework defined by Cardellini et al. [1] and propose an ILP model for the optimal allocation and a second ILP model as a heuristic which approximates the network energy consumption. The heuristic reduces the complexity of the optimization to a non-quadratic assignment problem but leads to worse results in terms of network energy consumption. In our work, we extend both models to allow the optimization of the allocation of multiple workflows simultaneously. This removes the bias towards already allocated workflows when allocating them one by one. In addition, we add capabilities of nodes and tasks to avoid e.g., mapping a sensor-reading task onto a non-sensor device.

In contrast to our ILP model, designed for an edge infrastructure, Skarlat et al. [20] define an ILP for a cloud and fog computing infrastructure and application deadlines. Their ILP objective function reduces the cost of running services on the cloud by placing as many services as possible on the fog network. In addition to the ILP model, Skarlat et al. use and evaluate an implementation of the GA as a heuristic. They find that the GA utilizes less fog resources than the optimal solution, which leads to an increase in cost of about 2.5. This is similar to the findings of You et al. [22] who compare the GA, simulated annealing and PSO for task offloading in edge computing networks. Their model focuses on the task execution delay and computation performance and penalizes higher energy consumption. They identify that in comparison to the other two algorithms, PSO converges faster, finds a solution with a lower energy consumption and has a lower task execution delay especially for a large number of nodes.

In addition to well-known heuristics, we also focus on machine learning approaches like DRL. Gao et al. [8] propose a DRL approach to offload multiple workflows on edge servers and user equipment. They minimize the energy consumption and completion time of the workflows with a multi-agents deep deterministic policy gradient algorithm. This algorithm yields the best values and terminates as fastest in comparison to random offloading and DQN-based offloading. Zheng et al. [23] implement a DQN-based task offloading algorithm which focuses on minimizing the task offloading delay by balancing the offloading between multiple edge servers.

### 4 METHODOLOGY

In our methodology, we show how we use and implement the methods described in Section 2. First, we explain the extended ILP problem. Then, we propose parameters for PSO and use parts of...
We base our ILP problem on the work of Seeger et al. [19] and extend workflows with one task each in the upper rectangle and a network location of all tasks is not feasible anymore since the task with four available resources (example the node with four available resources). If it does so, the algorithm allocates the task with three required resources to the node with the lowest energy consumption (in our simplified example of a worst-case. We start with three simple workflows with one task each in the upper rectangle and a network of three nodes below. The required and available resources of the tasks and nodes are shown as the number of squares. Tasks cannot be split across multiple nodes. We assume that the node with four available resources has the lowest energy consumption.

If we start allocating the task from left to right (one by one), the algorithm allocates the task with three required resources to the node with the lowest energy consumption (in our simplified example the node with four available resources). If it does so, the allocation of all tasks is not feasible anymore since the task with four required resources can only be placed on the node with four available resources. However, this node would be blocked by the task with three resources. If we solve the ILP problem for all workflows simultaneously, we avoid the scenario described in Figure 3.

The following equations describe our ILP model:

\[
\min F_{\text{total}}
\]  

subject to:

\[
\forall w \in W : \forall t_1, t_2 \in w : \forall n_1, n_2 \in N : Y[t_1, t_2, n_1, n_2] \leq X[t_1, n_1]
\]  

\[
\forall w \in W : \forall t_1, t_2 \in w : \forall n_1, n_2 \in N : Y[t_1, t_2, n_1, n_2] \leq X[t_2, n_2]
\]  

\[
\forall w \in W : \forall t_1, t_2 \in w : \forall n_1, n_2 \in N : Y[t_1, t_2, n_1, n_2] \geq X[t_1, n_1] + X[t_2, n_2] - 1
\]  

\[
\forall w \in W : \forall t \in w : \sum_{n \in N} X[t, n] = 1
\]  

\[
\forall w \in W : \forall t \in w : \forall n \in N : X[t, n] \leq F[t, n]
\]  

\[
\forall n \in N : \sum_{w \in W} \sum_{t \in w} X[t, n] \times R^n_{\text{RAM}} \leq R^n_{\text{RAM}}
\]  

\[
\forall n \in N : \sum_{w \in W} \sum_{t \in w} X[t, n] \times R^n_{\text{CPU}} \leq R^n_{\text{CPU}}
\]  

\[
\forall n \in N : \sum_{w \in W} \sum_{t \in w} X[t, n] \times R^n_{\text{Storage}} \leq R^n_{\text{Storage}}
\]  

\[
\sum_{w \in W} \sum_{t \in w} \sum_{n \in N} C_n \ast (S_t/P_n) \ast X[t, n] \leq E_{\text{device}}
\]  

\[
\sum_{w \in W} \sum_{t_1, t_2 \in w} \sum_{n_1, n_2 \in N} O_{t_1} \ast D_{n_1, n_2} \ast Y[t_1, t_2, n_1, n_2] \leq E_{\text{network}}
\]  

\[
E_{\text{device}} + E_{\text{network}} \leq E_{\text{total}}
\]

Our extended ILP model starts with the minimization of the total energy consumption as objective in Equation 6. We use X and Y as binary decision variables as shown in Table 2 whether a task is allocated on a node and whether two tasks on two nodes communicate with each other. If task \( t_1 \) is allocated to node \( n_1, X[t_1, n_1] = 1 \) otherwise 0. If both \( X[t_1, n_1] \) and \( X[t_2, n_2] \) are equal to 1, \( Y[t_1, t_2, n_1, n_2] = 1 \). Given only \( X[t_1, n_1] = 1 \), \( Y[t_1, t_2, n_1, n_2] \) could be zero or one depending on \( X[t_2, n_2] \). Therefore \( Y[t_1, t_2, n_1, n_2] \) is always smaller or equal to both \( X \). We use Equations 7 - 9 to define \( Y \). We use Equation 10 to allocate a task \( t \) only once on the network. This can be changed if we need redundancy of tasks as a QoS objective.

Equation 11 limits the placement of tasks to nodes which meet their required capabilities. If node \( n \) offers the capability that is needed by task \( t, F[t, n] = 1 \) or 0 otherwise. If \( F[t, n] = 0 \), the task cannot be allocated on node \( n \) and \( X[t, n] = 0 \).

The Equations 12 - 14 constrain the sum of the required resources of all tasks on a node to at maximum the available resources of that node. We differentiate between CPU cycles, RAM, and storage as resources for our edge infrastructure.

Equation 15 defines the energy consumption of the device depending on the computation size of the task, the processing power of the node and the energy consumption by CPU cycle. Equation 16 defines the network energy consumption depending on the task transmission output between two tasks and the energy cost of the path in between the nodes to which the tasks are allocated to. Equation 17 models the total energy consumption used as minimization objective in this ILP. Table 2 explains the variables used in the ILP model in more detail.

Since our model is the extended version of the model of Seeger et al. [19], we use their approach to create an ILP heuristic. The main difference in this work is the addition that allows us to allocate multiple workflows for the heuristic. Further, we added the resource and capability constraint to the ILP heuristic.

To solve the ILP problem, we use the Python library pulp in combination with the IBM CPLEX solver.

Next, we transfer the ILP model and the constraints to other approaches.

### 4.2 Particle Swarm Optimization

As described in Section 2, we use Deb’s approach [6] to implement the constraints of the ILP problem. As a result, \( f(x) = E_{\text{device}} + E_{\text{network}} \) from Equation 15 and 17. For \( f_{\text{worst}} \), we calculate the maximum total energy consumption for the nodes and the network as shown in Equation 18-20. In Equation 19 \( tasks \) is sum of the number of tasks over all workflows. In Equation 20 \( edges \) is the sum of all

\[
\sum_{w \in W} \sum_{t \in w} \sum_{n \in N} C_n \ast (S_t/P_n) \ast X[t, n] \leq E_{\text{device}}
\]

\[
\sum_{w \in W} \sum_{t_1, t_2 \in w} \sum_{n_1, n_2 \in N} O_{t_1} \ast D_{n_1, n_2} \ast Y[t_1, t_2, n_1, n_2] \leq E_{\text{network}}
\]

\[
E_{\text{device}} + E_{\text{network}} \leq E_{\text{total}}
\]
4.3 Deep Reinforcement Learning

For our DRL implementation, we use the Python library stable_baselines3 with a custom environment of the OpenAI gym library. We tested multiple models with different observation spaces and present the best performing DRL model. Our custom environment allocates tasks one by one. As a result, the action is the number of the node on which the task should be allocated. We used the same variables and calculations $S_t$, $P_n$, $E_{\text{device}}$ and $E_{\text{network}}$ as defined in the ILP problem. The observation space is a dictionary which consists of:

- A list of the normalized (between zero and one) computation size $S_t$ of each task $t$.
- A list of the processing power $P_n$ of each node $n$. The value is zero for invalid nodes.
- A list of values that show the increase in network energy consumption depending on the node on which the task is going to be allocated. If the node is invalid because of constraints, the value is two times the maximum network energy.

For the reward, we calculate the difference in total energy consumption $\Delta E_i$ between the current allocation and previous allocation and multiply it by -1 as shown in Equation 21 - 24. The multiplication of the energy cost with -1 is necessary because the DRL agent maximizes the reward. Therefore, the agent now aims to minimize the energy cost.

\[
\Delta E_0 = 0 \\
\Delta E_i = E^i_{\text{total}} - E^{i-1}_{\text{total}} \\
E^i_{\text{total}} = E^i_{\text{device}} + E^i_{\text{network}} \\
\text{reward} = -1 + \Delta E_i
\]

To avoid violating constraints, we use MaskablePPO in our implementation to only allow numbers of valid network nodes as actions.

5 EVALUATION

In this section, we show the results of the optimal and heuristic approaches in comparison to each other. In addition, we compare the run-time of each approach.

5.1 Experiment Setup

For our experiments, we use a machine running Debian bullseye, an Intel(R) Xeon(R) Silver 4116 CPU and 160 GB of RAM. We execute all experiments on the same machine. To compare simple task allocation problems with the optimal solution and the scalability of the heuristic algorithms, we split our experiments into two datasets. For the first dataset, we use simplified versions of the ILP, the ILP Heuristic, PSO and DRL where we omit the capability,
CPU and storage constraints. For the second dataset, we include all constraints but omit the optimal results of the ILP due to the computational complexity and time consumption.

We implemented a Python module which generates an example with the parameters workflow length, workflow width and workflow count. An example consists of one or multiple workflows and a network. The network is scaled with the number of tasks to allow a feasible solution. The parameters allow a pseudo-random workflow structure with some degrees of freedom e.g., in total number of tasks. Each example is identified by a number. We increase the length, width and workflow count for each example as shown in Figure 4 for both datasets.

We trained the DRL model with one million episodes for the simplified DRL version (first dataset) and with 80 million episodes for the full version (second dataset).

We solve the ILP problem once for all examples of the first dataset. To calculate a mean value and a standard deviation, we execute the ILP heuristic, PSO and DRL ten times on each example for both datasets. We sort the results of the examples in the next sections by the total number of tasks.

5.2 Comparison of Optimal and Heuristic Algorithms

Figures 5a and 5b show the results of the first dataset. The x-axis shows the total number of tasks and the ID of the examples. The y-axis shows the cost which is the total energy consumption of the devices and the network. The ILP yields the optimal value. Since the other approaches only approximate these optima, their results are worse than or equal to the ILP results.

In our experiments, the ILP heuristic and DRL are deterministic. Thus, we only include the standard deviation of PSO in the figures. Figure 5 shows in the simplified version, overall, PSO and DRL outperform the ILP heuristic. The best approach is PSO. However, the PSO algorithm yields worse results for more than 25 tasks in comparison to DRL. For a higher number of tasks, the best approximation is provided by the DRL. In addition, PSO is the only algorithm which can terminate with an infeasible solution and does so for example 34 with 35 tasks.

With increasing problem size, the ILP becomes infeasible. Due to this, we were not able to include the optimal solution of example 27, 28, 33, 34 and 47 despite running the solver for longer than two weeks. We show the time usage over all examples in Figure 6a. The maximum required time to solve an ILP problem is example 21 with

![Figure 4: Metadata of each example for the first dataset.](image-url)
5.3 Scalability Analysis

To check the performance of each algorithm for larger example sizes and in their extended version, we compare them using the second dataset. Since the dataset includes 120 examples and larger problem sizes, we omit the extended version of the ILP. The overall results are similar to the first dataset. The extended PSO algorithm yields the best results for less than 34 tasks. For over 34 tasks, DRL results in the lowest energy consumption. Overall, the ILP heuristic yields the highest energy consumption for all but two small examples with 4 and 5 tasks. The distribution of the execution time is comparable with the first dataset with a maximum execution time of 2.07 s for the extended ILP Heuristic, 360.70 s for the extended PSO algorithm and 0.99 s for the extended DRL model.

For examples with over 20 tasks, we notice an increase in failed attempts for PSO as shown in Figure 7. A failed attempt is an execution which terminates without finding a feasible solution. We ran each example ten times. For over 60 tasks, PSO failed ten attempts and was not able to find any feasible solution.

6 CONCLUSION

In this work, we extend the existing ILP problem definition of Seeger et al. [19] to minimize the energy consumption of the task allocation problem with multiple workflows. We propose a PSO and DRL approach which achieve median optimality gaps of 7.7 % and 35.9 %. DRL achieves the lowest upper bound of the optimality gap with 115.9 % in comparison to 288.7 % for the ILP heuristic and 172.8 % for PSO. Our results show that PSO sometimes fails to allocate tasks past a size of 25-34 tasks. Furthermore, DRL generally scales better for larger problem sizes. In addition, our results indicate that the extended PSO algorithm is unreliable for the allocation of more than 20 tasks and unusable for the allocation of more than 60 tasks. Our measurements determine the extended DRL model as fastest algorithm with an execution time of under 1 s in comparison to 2 s for the extended ILP heuristic and 360 s for the extended PSO. As a result, our results show that PSO is more suitable for smaller and DRL for larger sized task allocation problems.

As future work, we plan to analyze the trade-off between speed and optimality in ILP problems when adding a bias towards previously allocated tasks. In addition, we identify possible future research in the direction of heterogeneous networks e.g., by introducing a cloud connection or the limitations and opportunities of 5G and TSN.

ACKNOWLEDGMENTS

This work has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 957218 (Project IntellIoT).

REFERENCES


